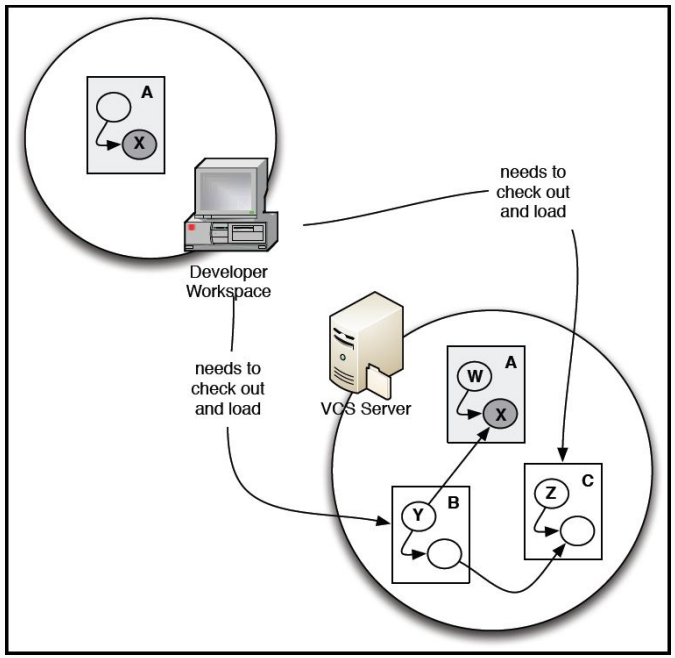
Taming Large Models with Hawk and NeoEMF

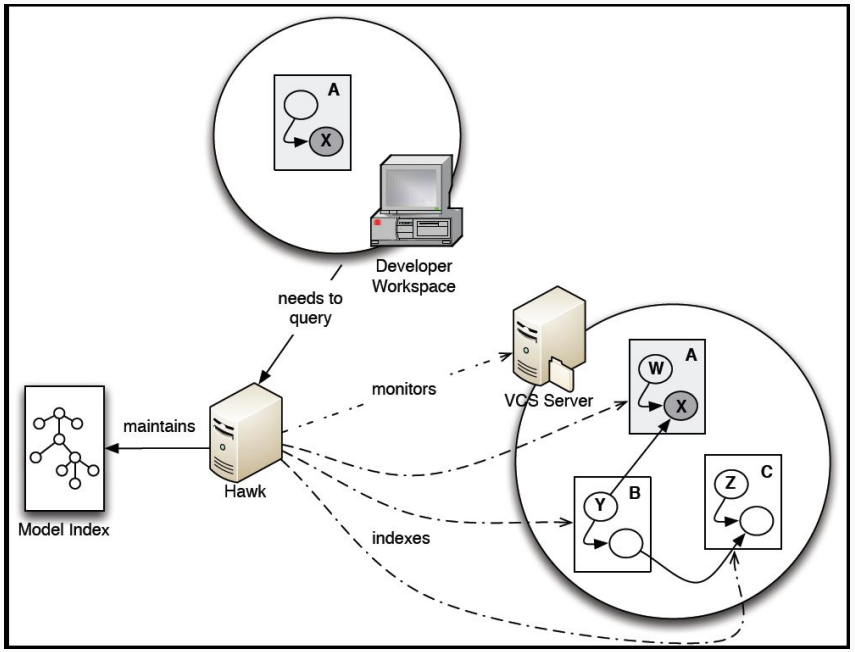
A. García-Domínguez, D. S. Kolovos, K. Barmpis, G. Daniel, G. Sunyé

MoDELS'2018, 14–19 October 2018

Hawk

Hawk: indexing for fast querying over fragment collections





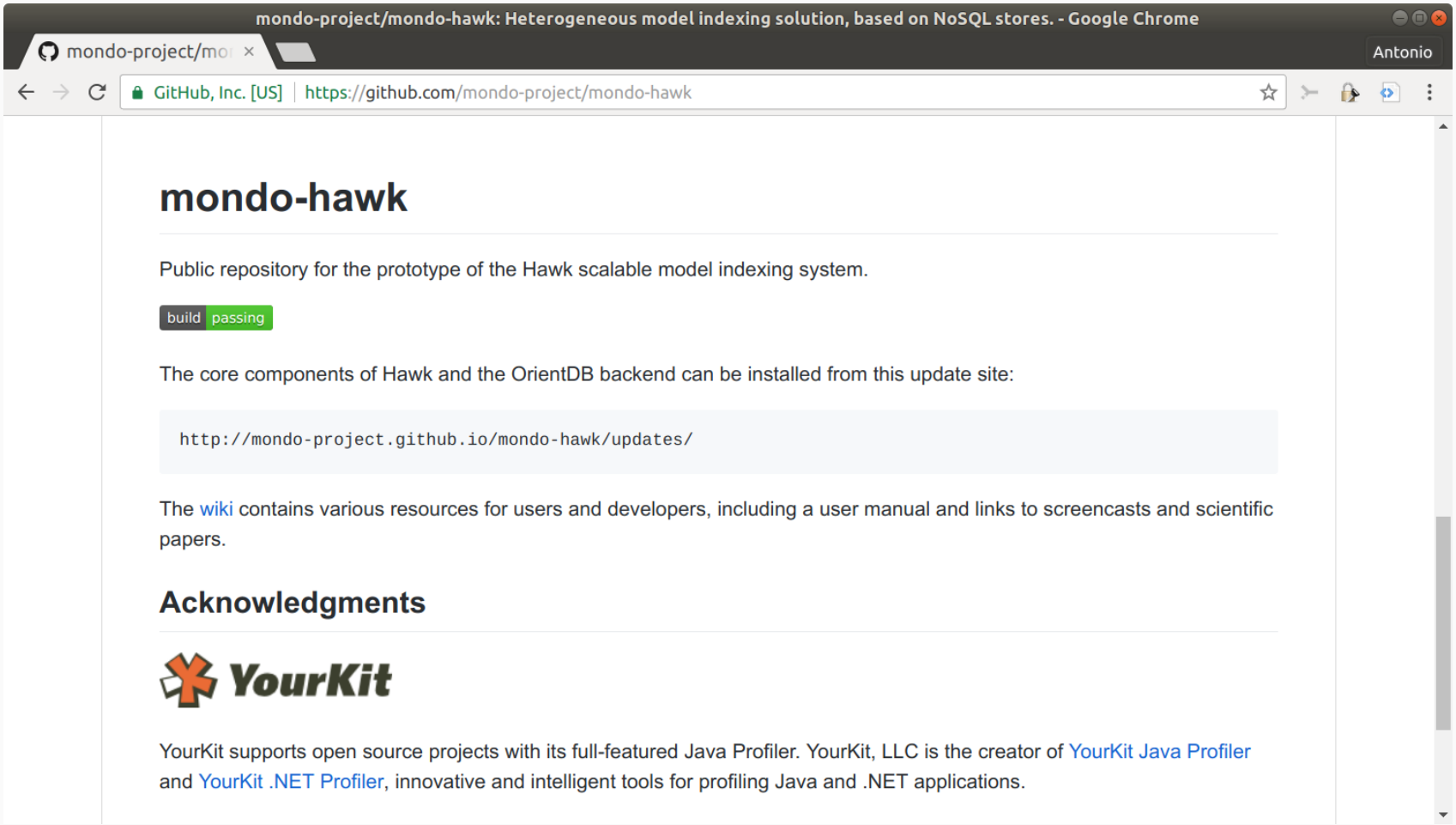
Usual approach

1. Check out **all** files from VCS
2. Load fragments into memory
3. Run query (might go over all fragments)

With Hawk

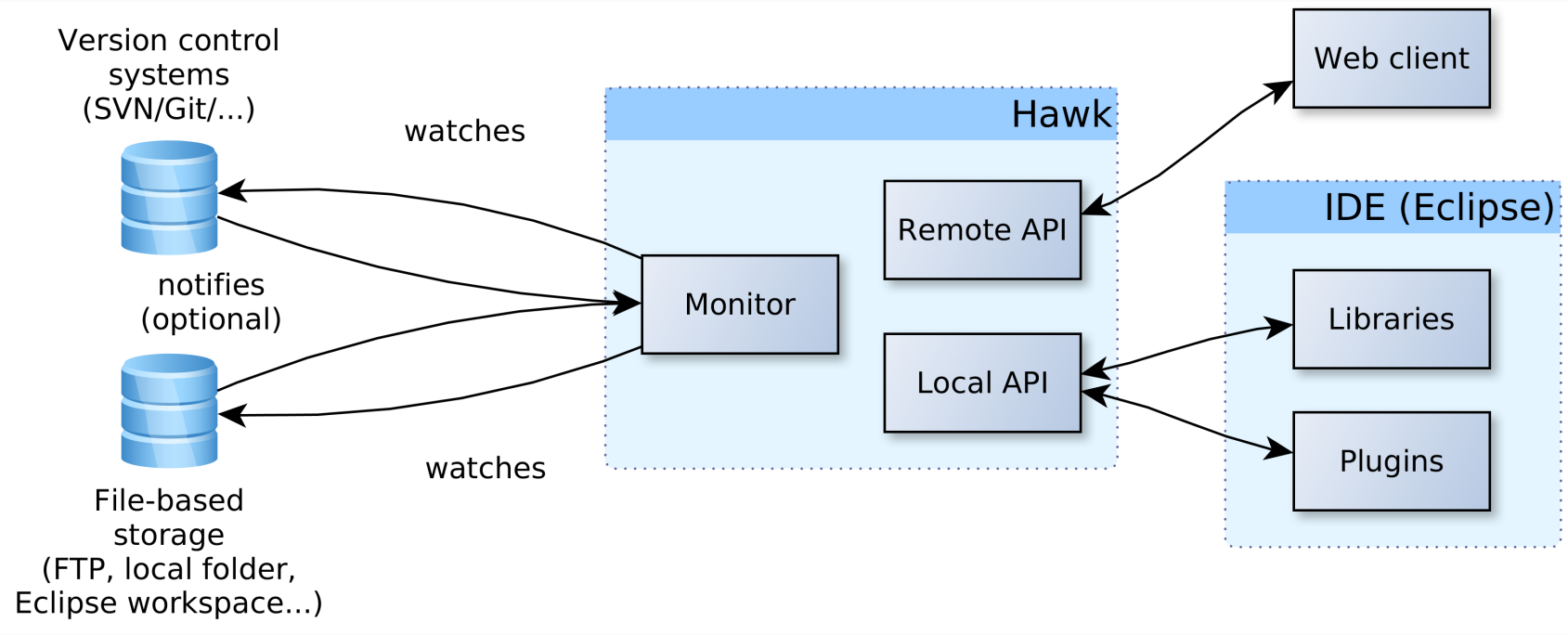
1. Hawk watches VCS, indexes
2. User queries Hawk over WS
3. Hawk runs query through NoSQL database efficiently
4. Hawk replies with result

Hawk: project website



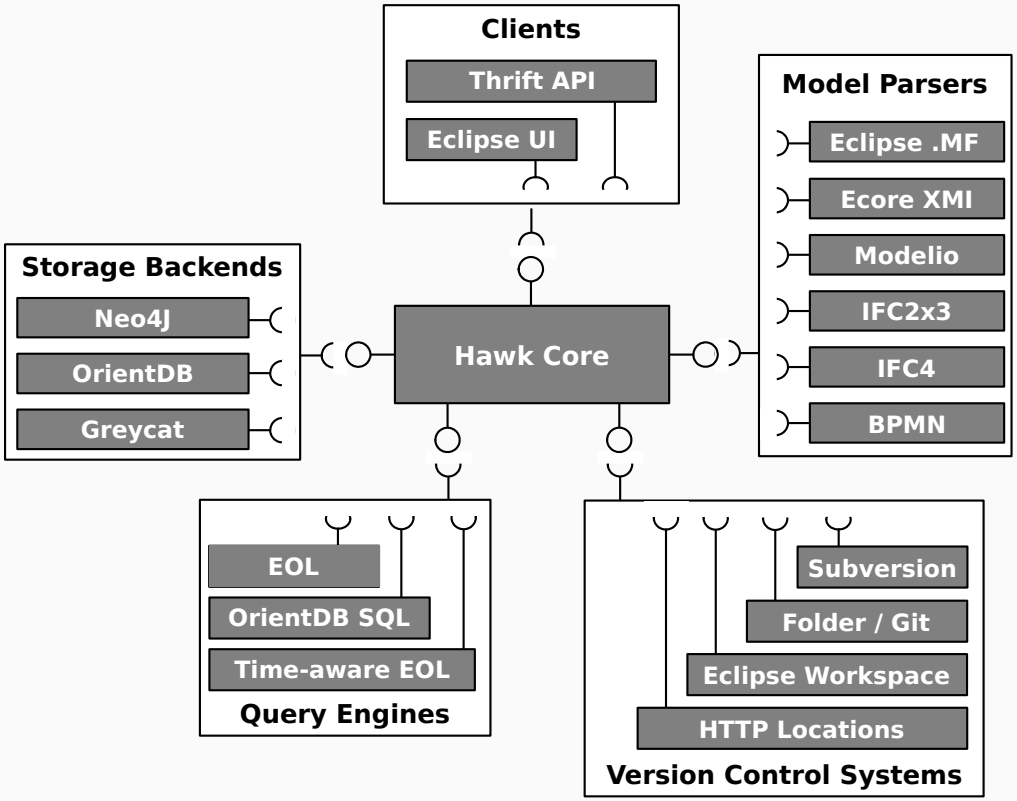
- `https://github.com/mondo-project/mondo-hawk`
- Recently accepted as Eclipse Incubator project (moving soon!)

Hawk: deployment



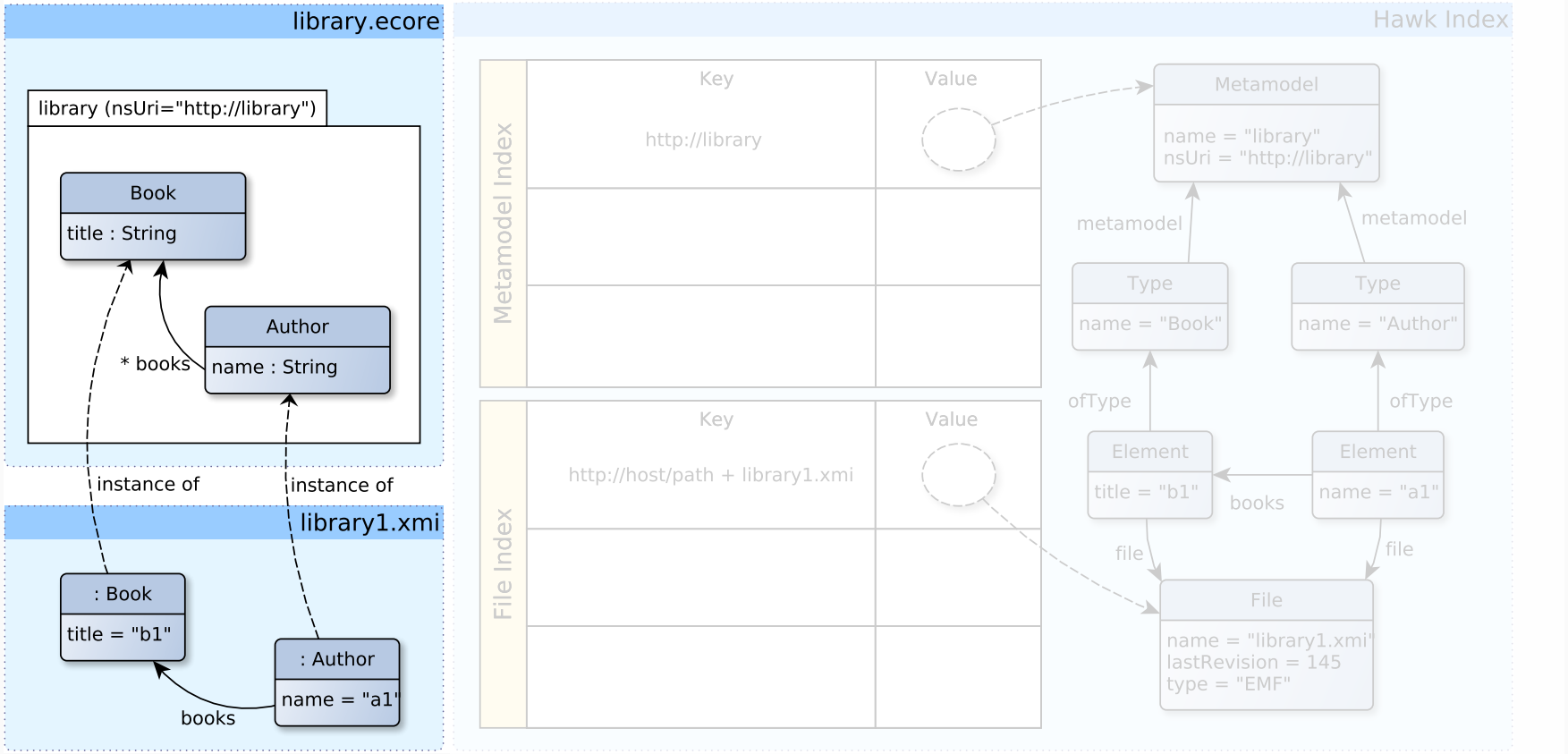
- Hawk can run as Eclipse plug-in, Java library, or network service
- We can have it watch over various types of locations:
  - Version control systems (SVN/Git repositories)
  - File stores (local folders, Eclipse workspaces, HTTP locations)

Hawk: component-based architecture



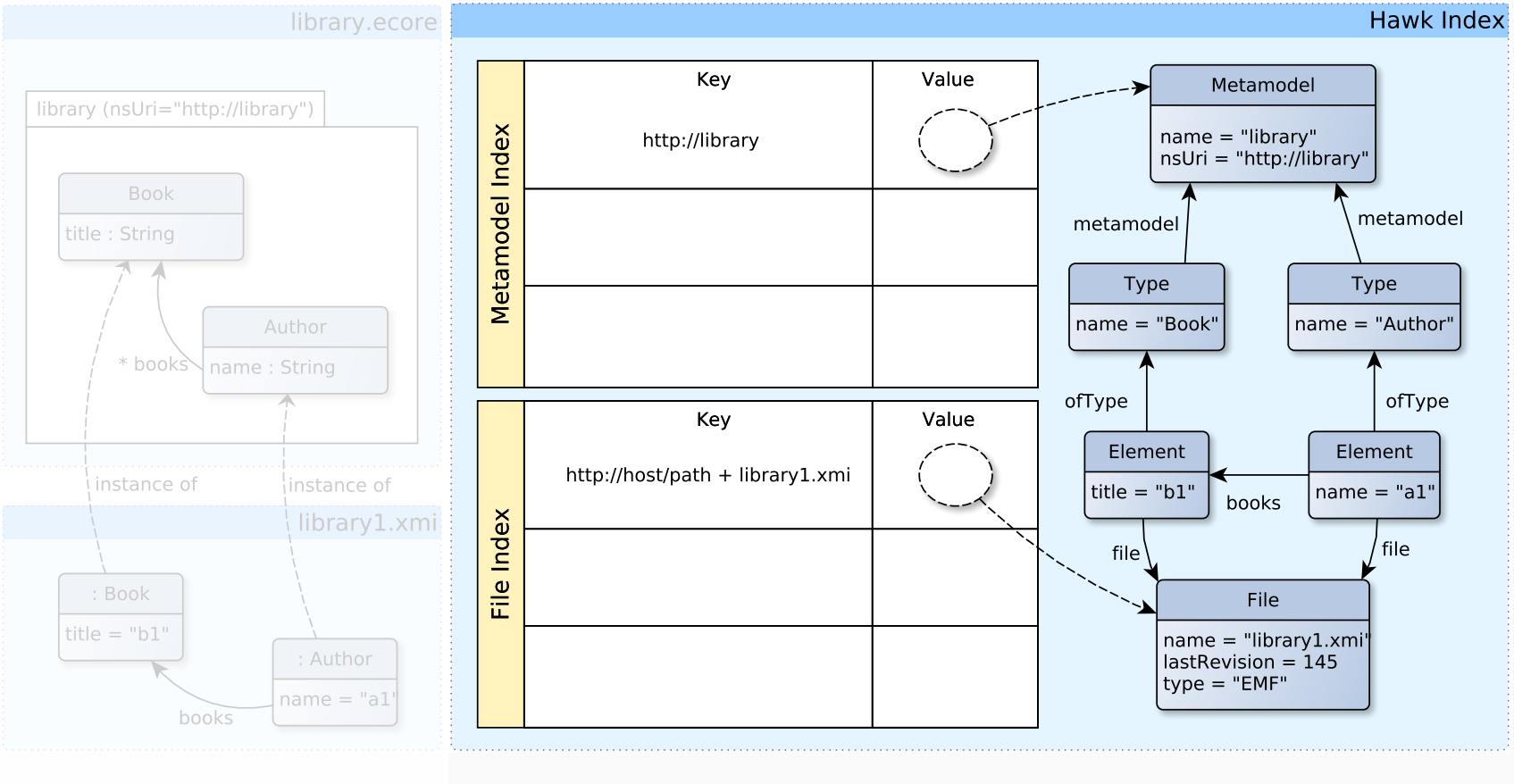
- Core: incremental graph updating + component interfaces
- Backends: Neo4j (fastest), OrientDB (multi-master), Greycat
- Clients: Eclipse GUI, cross-language Apache Thrift web services
- Query engines: Epsilon Object/Pattern Languages, OrientDB SQL
- Model parsers: EMF/Modelio models, Eclipse plug-in manifests...

Hawk: example for a library model



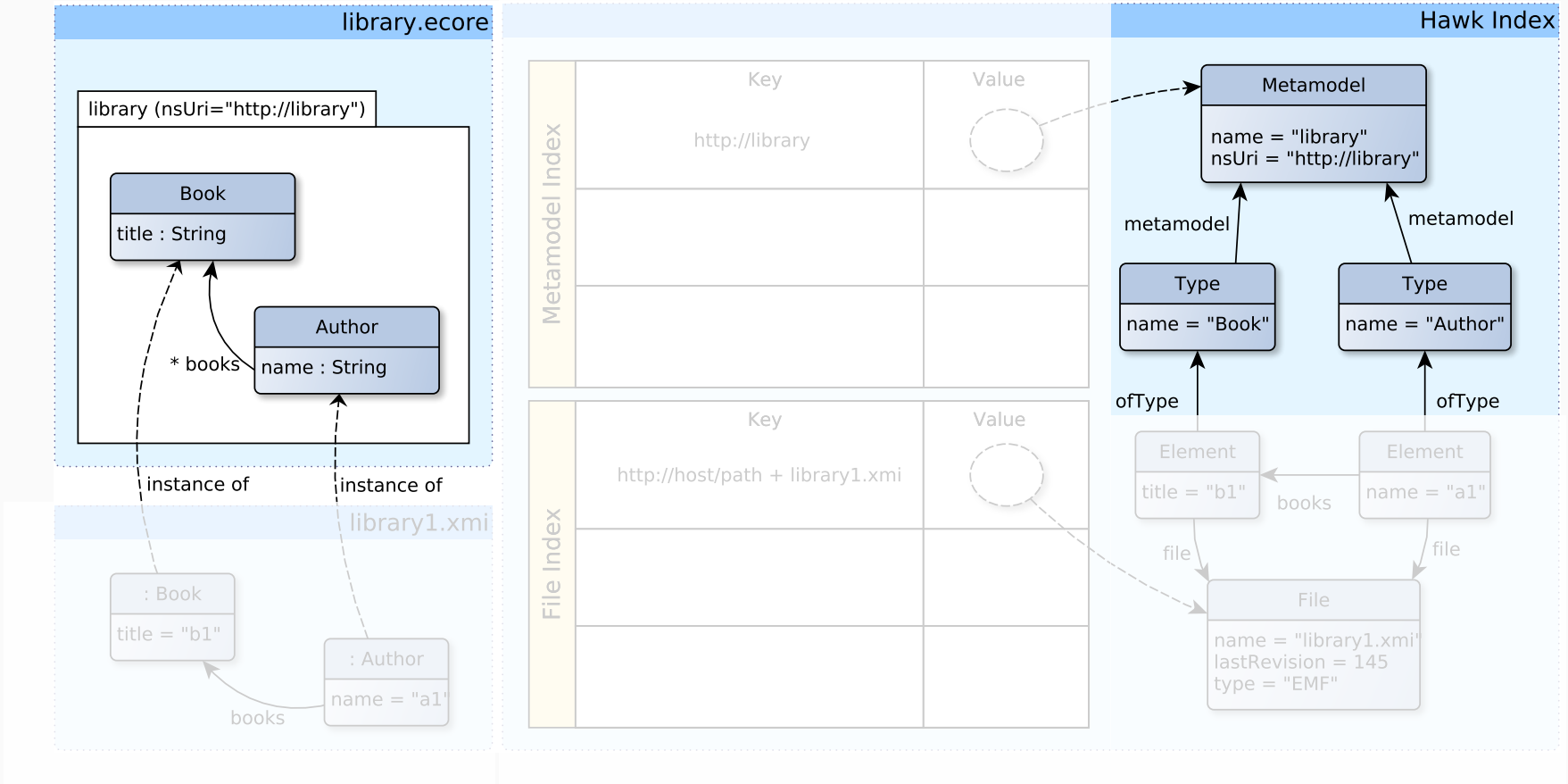
We go from these model files...

Hawk: example for a library model



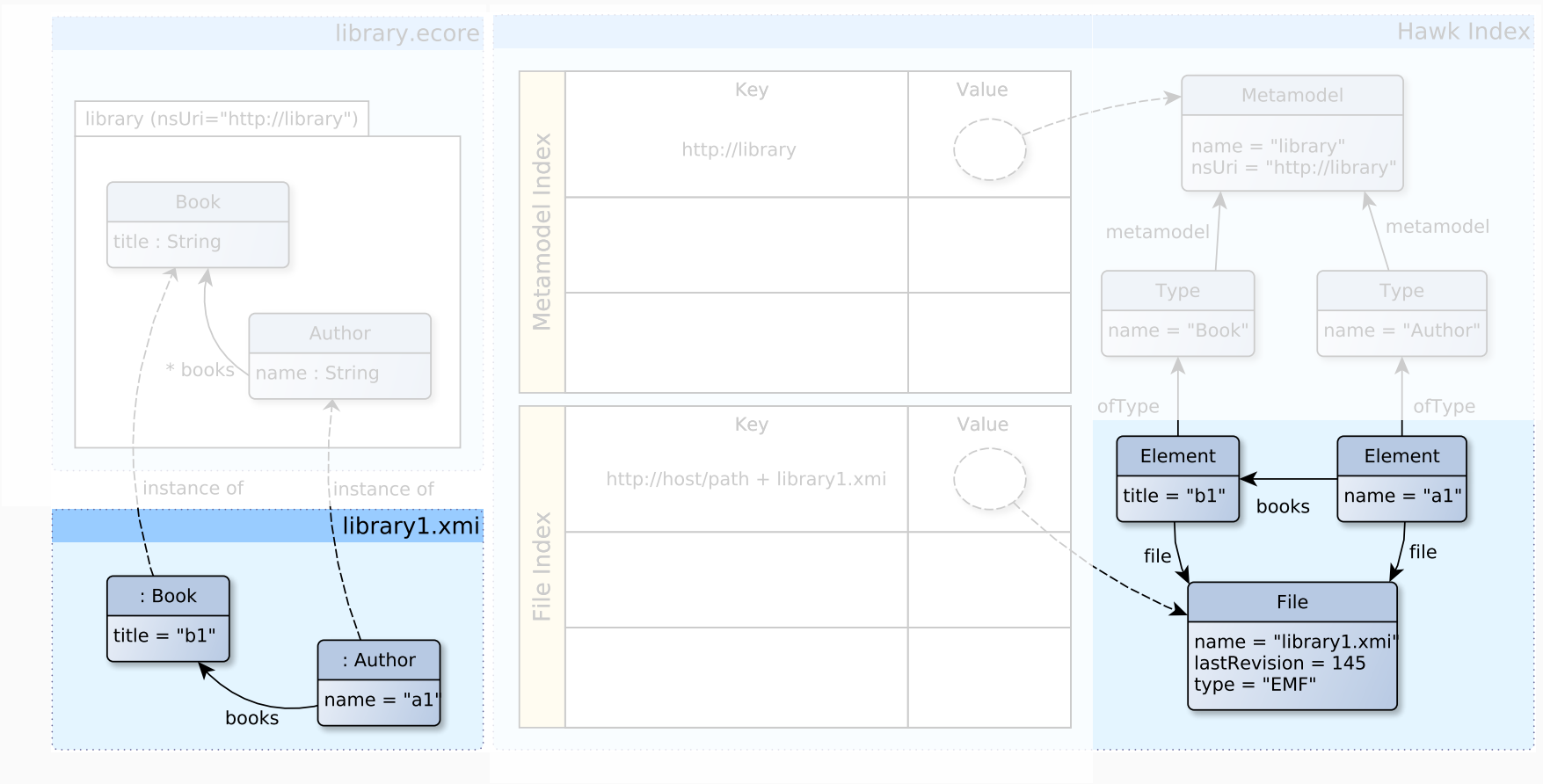
... to these NoSQL graphs.

Hawk: example for a library model



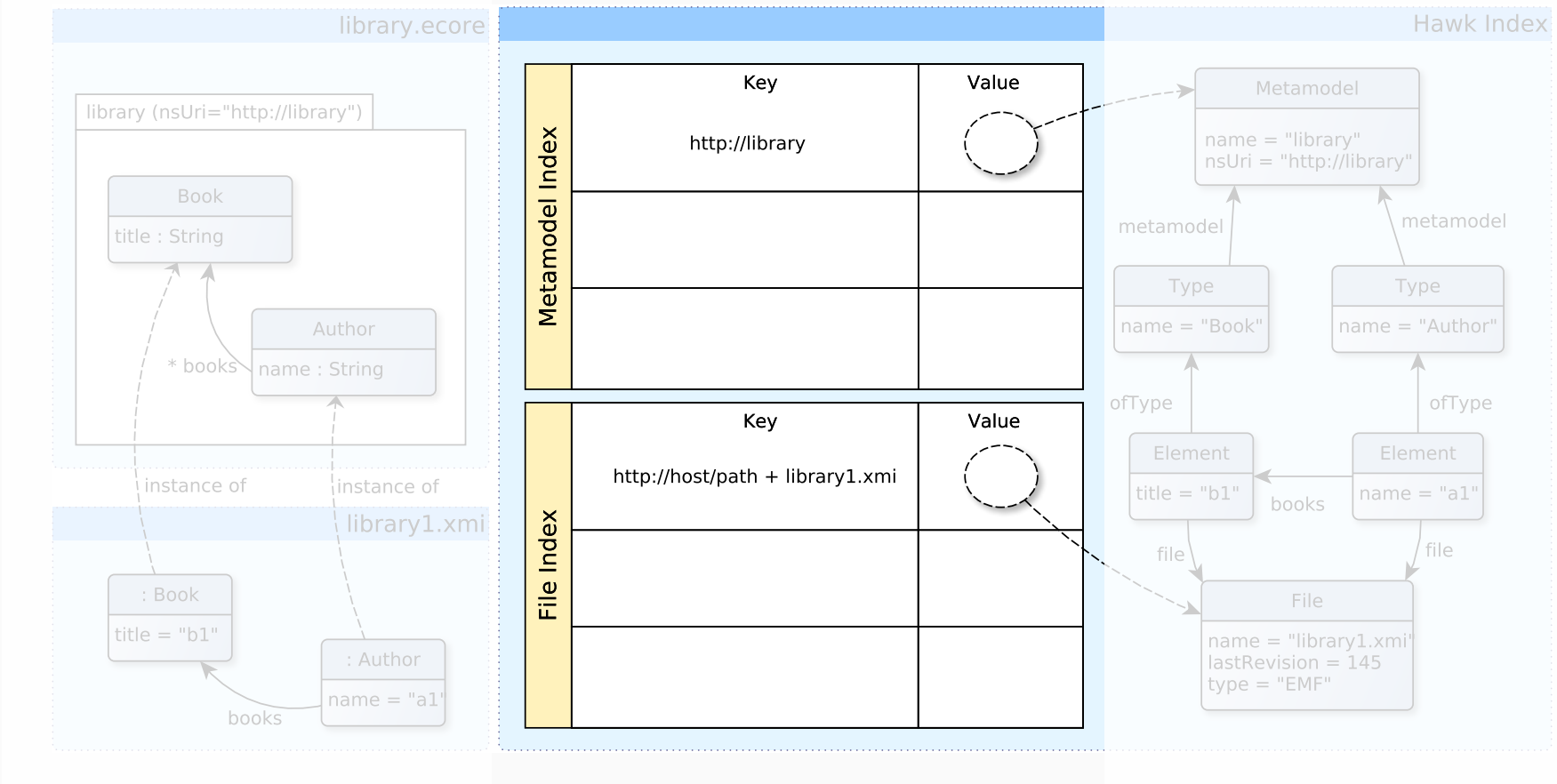
- Ecore packages → metamodel nodes
- Ecore classes → type nodes

Hawk: example for a library model



- Physical files → file nodes
- Model elements → element nodes

Hawk: example for a library model



- MM index: package URI $\rightarrow$ metamodel node
- File index: file path $\rightarrow$ file node
- Users can define custom indices by attribute/expression

Demo time!

Let's index a Java model and find singletons.

Hawk: indexed attributes

Finding a type by its name

```
return TypeDeclaration.all.selectOne(td |
  td.name.identifier = 'IdeInputFileObject');
```

This normally involves...

1. Iterating over all types
2. Following the "name" reference
3. Comparing the name

Hawk can replace this with a lookup

We only need to tell it to index "SimpleName.identifier":

```
return SimpleName.all
  .select(sn | sn.identifier='IdeInputFileObject')
  .eContainer.select(c|c.isKindOf(TypeDeclaration));
```

Hawk: derived attributes

Original query for finding singletons

```
return TypeDeclaration.all.select(td | td.bodyDeclarations.exists(
  md:MethodDeclaration | md.returnType.isTypeOf(SimpleType)
    and md.returnType.name.fullyQualifiedName
      = td.name.fullyQualifiedName
    and md.modifiers.exists(mod:Modifier | mod.public = true)
    and md.modifiers.exists(mod:Modifier | mod.static = true)
));
```

Can we do it faster?

- Checking if a method is public or static requires traversing references
- Same goes for checking if it returns an instance of itself
- In Hawk, we can precompute this
- When files change, only the affected values are recomputed

Hawk: use of derived attributes as precomputed values

Original query

```
return TypeDeclaration.all.select(td | td.bodyDeclarations.exists(
  md:MethodDeclaration | md.returnType.isTypeOf(SimpleType)
    and md.returnType.name.fullyQualifiedName
      = td.name.fullyQualifiedName
    and md.modifiers.exists(mod:Modifier | mod.public = true)
    and md.modifiers.exists(mod:Modifier | mod.static = true)
));
```

Changed to use derived attributes on MethodDeclaration

```
return TypeDeclaration.all.select(td|
  td.bodyDeclarations.exists(md:MethodDeclaration |
    md.isPublic = true
    and md.isStatic = true
    and md.isSameReturnType = true));
```

Hawk: derived attributes are also indexed

Revised query

```
return TypeDeclaration.all.select(td|
  td.bodyDeclarations.exists(md:MethodDeclaration |
    md.isPublic = true
    and md.isStatic = true
    and md.isSameReturnType = true));
```

Can we do it faster?

- Right now, we need to go through *all* type declarations and then filter by methods
- What if we go from the methods to the types instead?
- In Hawk, top-level selects can replace iteration with lookups when using derived attributes

Hawk: use of derived attributes as index keys

Previous query

```
return TypeDeclaration.all.select(td|
  td.bodyDeclarations.exists(md:MethodDeclaration |
    md.isPublic = true
    and md.isStatic = true
    and md.isSameReturnType = true));
```

Revised to use index, by using derived attributes at the top level

```
return MethodDeclaration.all.select(md |
  md.isPublic = true and md.isStatic = true
  and md.isSameReturnType = true
).collect( td | td.eContainer ).asSet;
```

Hawk: flagging singletons directly

Previous query

```
return MethodDeclaration.all.select(md |
  md.isPublic = true and md.isStatic = true
  and md.isSameReturnType = true
).collect( td | td.eContainer ).asSet;
```

Can we do it faster?

- We could just flag types that are singletons
- This derived attribute might be less reusable, however

Hawk: final query for finding singletons

Previous query

```
return MethodDeclaration.all.select(md |
   md.isPublic = true and md.isStatic = true
   and md.isSameReturnType = true
).collect( td | td.eContainer ).asSet;
```

Final query

```
return TypeDeclaration.all.select(td | td.isSingleton = true);
```

Demo time!

This time, we will show how to use indexed and derived attributes.

Derived edges

Toy example: Person metamodel

- Person metamodel, with "parents" references.
- We want to be able to quickly find siblings, grandparents, uncles/aunts, cousins, second-cousins, ancestors...
- We can precompute this in Hawk with derived edges

Derivation logic for "grandparents"

We need a flat list and not a list of lists, so we use "flatten":

```
return self.parents.parents.flatten;
```

Derivation logic for "siblings"

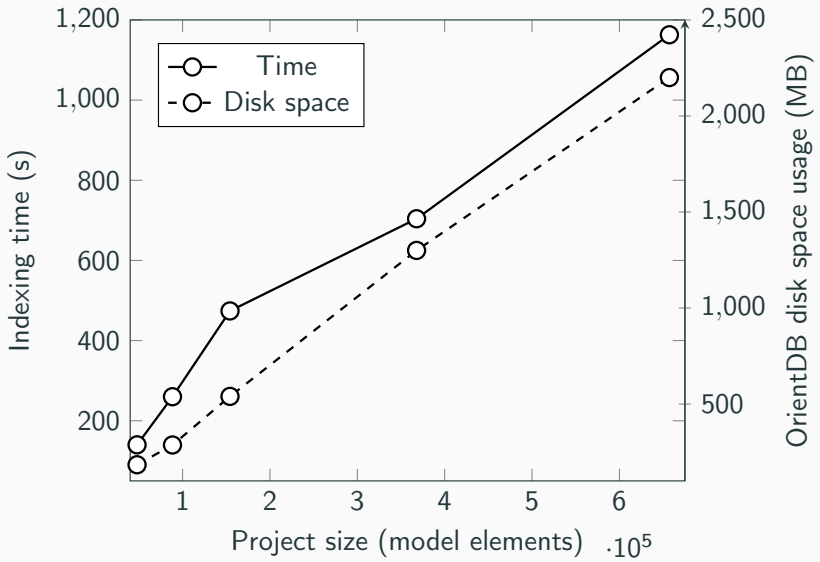
We can travel references in reverse with "revRefNav_name":

```
return self.parents.revRefNav_parents.flatten.excluding(self);
```

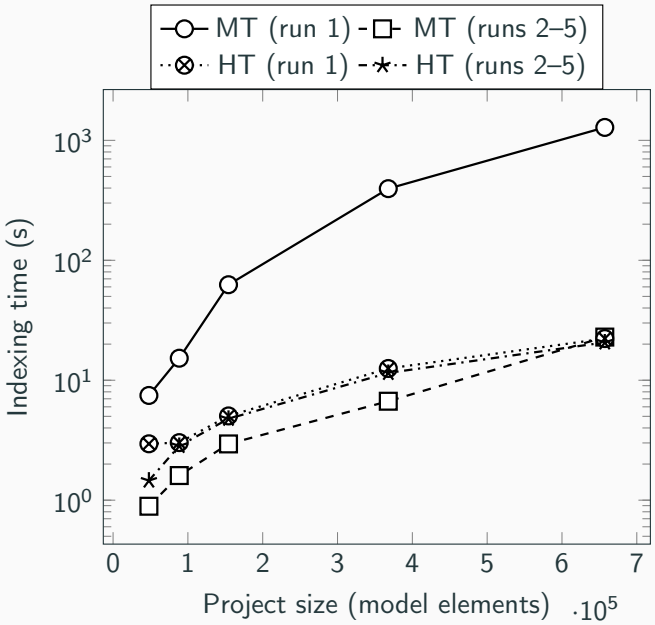
Last demo for Hawk.

We will show derived edges this time.

Hawk: integration into SOFTEAM Constellation [GDBK+16]



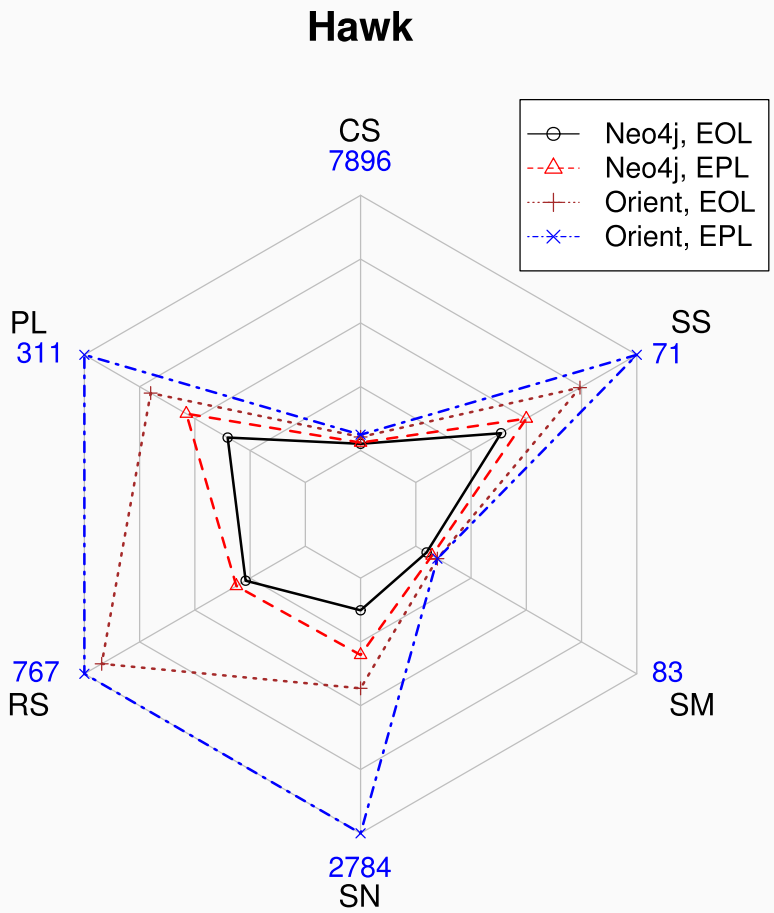
Indexing times and index sizes (OrientDB backend)

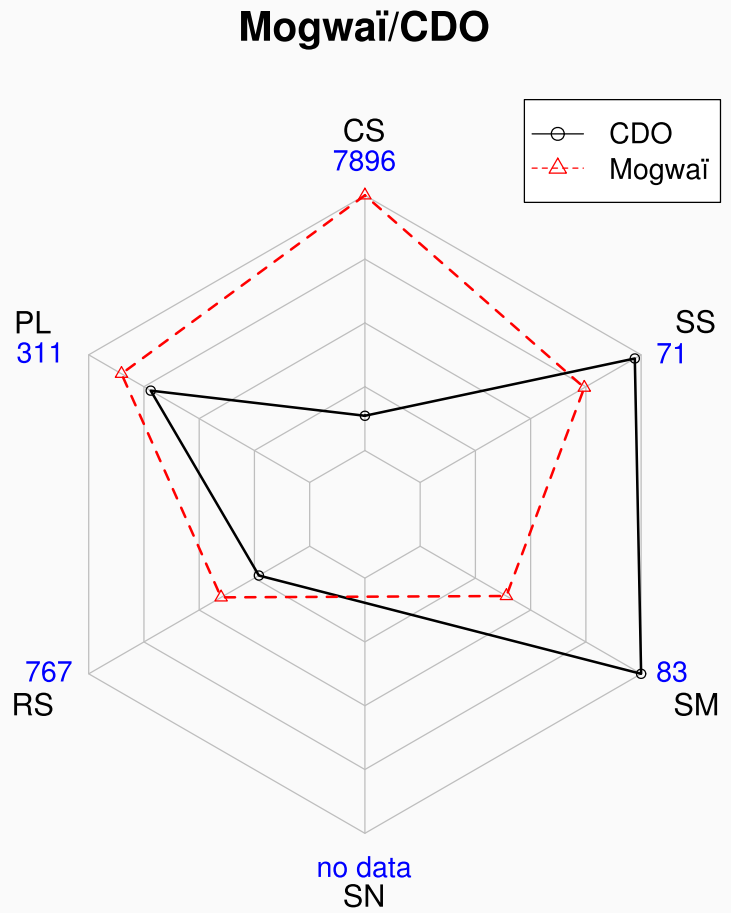


Code generation times: Modelio (MT), Hawk (HT)

- Constellation: collaboration platform over Modelio models
- SOFTEAM needed search, couldn't change persistence
- Integrated Hawk as a library: initial indexing cost quickly paid off

Hawk: stress-testing remote query APIs [GDBK+17]





- Included CDO, Hawk, Mogwaï and ranged 1–64 clients
- Reverse reference navigation was crucial in having the SN Train Benchmark [SIRV17] query run quickly

Hawk: time-aware queries over versioned models [GDBP18]

```
var rs = RewardTableRow.latest.all.collect(row | row.getRewardShifts()).flatten();
return Sequence { rs.min(), rs.max(), rs.average() };

operation RewardTableRow getRewardShifts(): Sequence {
  var v = self.versions;
  if (v.size <= 1) { return Sequence {}; }
  else { return v.subList(0, v.size - 1).collect(v | v.value - v.prev.value); }
}
operation Sequence average() { return self.sum() / self.size(); }
```

- Extended Hawk with Greycat temporal graph support and time-aware query engine / updater components
- Can index entire Subversion-based history of a model, and ask things about its history through a new set of time-aware primitives
- Above query is finding descriptive stats for reward table shifts in a models@run.time system
- Presenting this work at 14:00 (MRT'18) — hope to see you there!

Hawk: summing up

So far...

- Hawk is good for indexing an existing collection of model files
- You can efficiently answer queries from the index
- Indexed/derived features can be used to speed up queries

Ideas in the roadmap

- Extensible UI, covering differences in options for components
- Horizontal scaling (a flock of Hawks?)
- Web UI based on Thrift API
- More backends! (Triple stores? Neo4j 3.x? MapDB?)
- Better Git connector (JGit-based)
- Extending EOL with Linear Temporal Logic
- Visualizations based on time-aware queries

Feedback and contributions are welcome!

References i

Antonio Garcia-Dominguez, Konstantinos Barmpis, Dimitrios S Kolovos, Marcos Aurelio Almeida da Silva, Antonin Abherve, and Alessandra Bagnato.
Integration of a graph-based model indexer in commercial modelling tools.
In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 340–350, Saint Malo, France, 2016. ACM Press.

Antonio Garcia-Dominguez, Konstantinos Barmpis, Dimitrios S. Kolovos, Ran Wei, and Richard F. Paige.
Stress-testing remote model querying APIs for relational and graph-based stores.
Software & Systems Modeling, pages 1–29, June 2017.

References ii

A. García-Domínguez, N. Bencomo, and Luis H. García Paucar. **Reflecting on the past and the present with temporal graph-based models.** In *Proceedings of the 13th International Workshop on Models@run.time*, 2018. To be published.

Gábor Szárnyas, Benedek Izsó, István Ráth, and Dániel Varró. **The Train Benchmark: cross-technology performance evaluation of continuous model queries.** *Software & Systems Modeling*, January 2017.